CS 267 Applications of Parallel Computers

Lecture 9:

Sources of Parallelism and Locality

David H. Bailey

based on previous notes by Jim Demmel and Dave Culler

http://www.nersc.gov/~dhbailey/cs267

Recap: Parallel Models and Machines

- **Machine models**
  - shared memory
  - distributed memory
  - SIMD

  Programming models
  - threads
  - message passing
  - data parallel
  - shared address space

- **Steps in creating a parallel program**
  - decomposition
  - assignment
  - orchestration
  - mapping

- **Performance in parallel programs**
  - try to minimize performance loss from
    - load imbalance
    - communication
    - synchronization
    - extra work

Outline

- Simulation models
- A model problem: sharks and fish
- Discrete event systems
- Particle systems
- Lumped systems
- Ordinary Differential Equations (ODEs)
- Next time: Partial Different Equations (PDEs)

Simulation Models
and
A Simple Example

Sources of Parallelism and Locality in Simulation

- **Real world problems have parallelism and locality:**
  - **Many objects operate independently of others.**
  - **Objects often depend much more on nearby than distant objects.**
  - **Dependence on distant objects can often be simplified.**
- **Scientific models may introduce more parallelism:**
  - **When a continuous problem is discretized, temporal domain dependencies are generally limited to adjacent time steps.**
  - **Far-field effects may be ignored or approximated in many cases.**
- **Many problems exhibit parallelism at multiple levels**
  - **Example: circuits can be simulated at many levels, and within each there may be parallelism within and between subcircuits.**

Basic Kinds of Simulation

- Discrete event systems:
  - Examples: ”Game of Life”, timing-level simulation for circuits.
- Particle systems:
  - Examples: billiard balls, semiconductor device simulation, galaxies.
- Lumped variables depending on continuous parameters:
  - ODEs, e.g., circuit simulation (Spice), structural mechanics, chemical kinetics.
- Continuous variables depending on continuous parameters:
  - PDEs, e.g., heat, elasticity, electrostatics.

- A given phenomenon can be modeled at multiple levels.
- Many simulations combine more than one of these modeling techniques.

A Model Problem: Sharks and Fish

- **Basic idea: sharks and fish living in an ocean**
  - rules for movement (discrete and continuous).
  - breeding, eating, and death.
  - forces in the ocean.
  - forces between sea creatures.
- **6 problems (S&F1 - S&F6)**
  - Different sets of rule, to illustrate different phenomena.
- **Available in Matlab, Threads, MPI, Split-C, Titanium, CMF, CMMD, pSather**
  - not all problems in all languages.
- **www.cs.berkeley.edu/~demmel/cs267/Sharks_and_Fish**

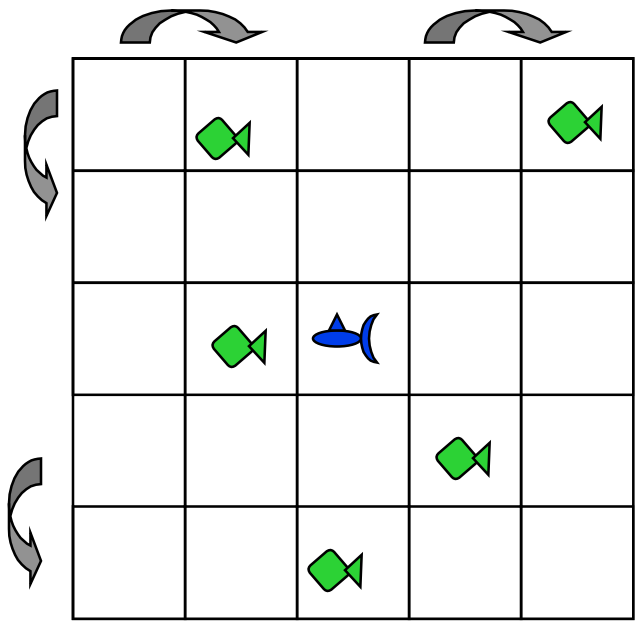
Discrete Event Systems

Discrete Event Systems

- **Systems are represented as:**
  - **finite set of variables.**
  - **each variable can take on a finite number of values.**
  - **the set of all variable values at a given time is called the state.**
  - **each variable is updated by computing a transition function depending on the other variables.**
- **System may be:**
  - **synchronous: at each discrete timestep evaluate all transition functions; also called a finite state machine.**
  - **asynchronous: transition functions are evaluated only if the inputs change, based on an "event" from another part of the system; also called event driven simulation.**
- **Example: functional level circuit simulation:**
  - **state is represented by a set of boolean variables (high & low voltages).**
  - **set of logical rules defining state transitions (and, or, etc.).**
  - **synchronous: only interested in state at clock ticks.**

Sharks and Fish as Discrete Event System

- Ocean modeled as a 2D toroidal grid.
- Each cell occupied by at most one sea creature.
- S&F 3, 4 and 5 are variations on this.



The Game of Life (Sharks and Fish 3)

- Fish only, no sharks.
- An new fish is born if
  - a cell is empty.
  - exactly 3 (of 8) neighbors contain fish.
- A fish dies (of overcrowding) if
  - cell contains a fish.
  - 4 or more neighboring cells are full.
- A fish dies (of loneliness) if
  - cell contains a fish.
  - less than 2 neighboring cells are full.
- Other configurations are stable.

Parallelism in Sharks and Fish

- **The simulation is synchronous**
  - **use two copies of the grid (old and new).**
  - **the value of each new grid cell depends only on 9 cells (itself plus 8 neighbors) in old grid.**
  - **simulation proceeds in timesteps, where each cell is updated at every timestep.**
- **Easy to parallelize using domain decomposition**

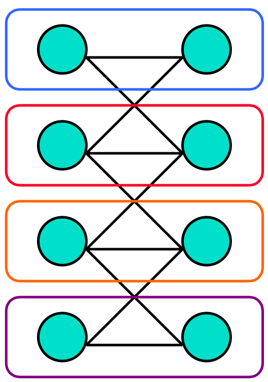
P1	P2	P3
P4	P5	P6
P7	P8	P9

```
Repeat
    compute locally to update local system
    barrier()
    exchange state info with neighbors
until done simulating
```

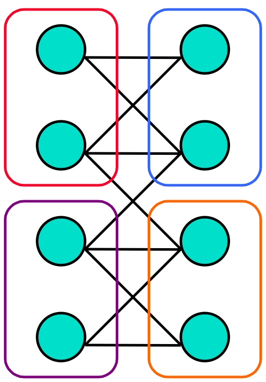
- **Locality is achieved by using large patches of the ocean**
  - **boundary values from neighboring patches are needed.**
- **If only cells next to occupied ones are visited (an optimization), then load balance is more difficult. The activities in this system are discrete events.**

Parallelism in Synchronous Circuit Simulation

- **Circuit is a graph made up of subcircuits connected by wires**
  - **Component simulations need to interact if they share a wire.**
  - **Data structure is irregular (graph).**
  - **Parallel algorithm is synchronous:**
    - **compute subcircuit outputs.**
    - **propagate outputs to other circuits.**
- **Graph partitioning assigns subgraphs to processors**
  - **Determines parallelism and locality.**
  - **Attempts to evenly distribute subgraphs to nodes (load balance).**
  - **Attempts to minimize edge crossing (minimize communication).**
  - **Nodes and edges may both be weighted by cost.**
  - **NP-complete to partition optimally, but many good heuristics exist (later lectures).**



edge crossings = 6



edge crossings = 10

Parallelism in Asynchronous Circuit Simulation

- **Synchronous simulations may waste time:**
  - **Simulate even when the inputs do not change, with little internal activity.**
  - **Activity varies widely across circuit.**
- **Asynchronous simulations update only when an event arrives from another component:**
  - **No global time steps, but individual events contain time stamp.**
  - **Example: Circuit simulation with delays (events are gates changing).**
  - **Example: Traffic simulation (events are cars changing lanes, etc.).**

Scheduling Asynchronous Circuit Simulation

- **Conservative:**
  - Only simulate up to (and including) the minimum time stamp of inputs.
  - May need deadlock detection if there are cycles in graph, or else "null messages".
  - Example: Pthor circuit simulator in Splash1 from Stanford.
- **Speculative:**
  - Assume no new inputs will arrive and keep simulating, instead of waiting.
  - May need to backup if assumption wrong.
  - Example: Parswec circuit simulator of Yelick/Wen.
  - Example: New RISC CPUs designs employ "speculative execution".
- **Optimizing load balance and locality is difficult:**
  - Locality means putting tightly coupled subcircuit on one processor.
  - Since "active" part of circuit likely to be in a tightly coupled subcircuit, this may be bad for load balance.

Particle Systems

Particle Systems

° **A particle system has**
- **a finite number of particles.**
- **moving in space according to Newton's Laws (i.e. $\mathbf{F} = m\mathbf{a}$).**
- **time is continuous.**

° **Examples:**
- **stars in space with laws of gravity.**
- **electron beam and ion beam semiconductor manufacturing.**
- **atoms in a molecule with electrostatic forces.**
- **neutrons in a fission reactor.**
- **cars on a freeway with Newton's laws plus model of driver and engine.**

° **Many simulations combine particle simulation techniques with some discrete event techniques (e.g., Sharks and Fish).**

Forces in Particle Systems

- **Force on each particle can be decomposed into near and far:**

  force = external_force + nearby_force + far_field_force

- **External force**
  - **ocean current to sharks and fish world (S&F 1).**
  - **externally imposed electric field in electron beam.**
- **Nearby force**
  - **sharks attracted to eat nearby fish (S&F 5).**
  - **balls on a billiard table bounce off of each other.**
  - **Van der Wals forces in fluid (1/r^6).**
- **Far-field force**
  - **fish attract other fish by gravity-like (1/r^2) force (S&F 2).**
  - **gravity, electrostatics, radiosity.**
  - **forces governed by elliptic PDE.**

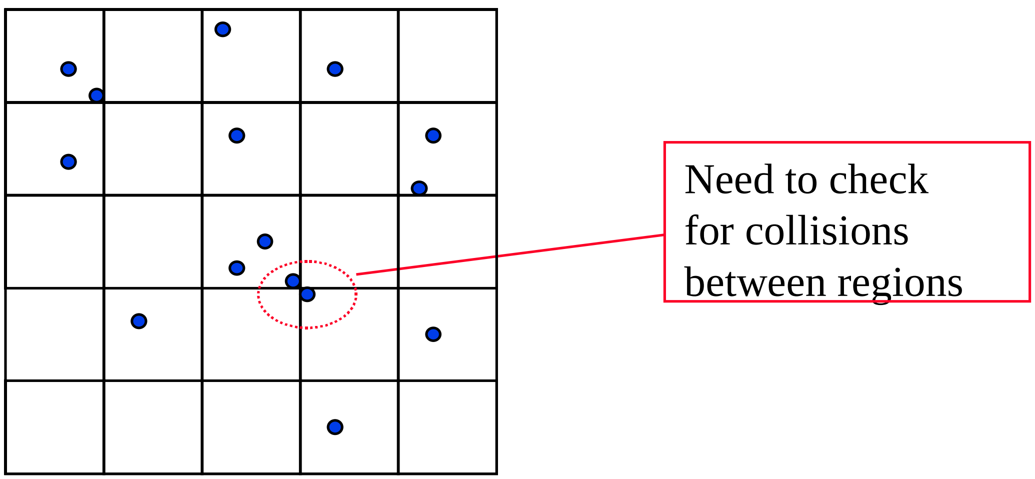
Parallelism in External Forces

- These are the simplest.
- The force on each particle is independent of other particles.
- Called “embarrassingly parallel”.

- Evenly distribute particles on processors
  - Any distribution works.
  - Locality is not an issue, no communication.
- For each particle on processor, apply the external force.

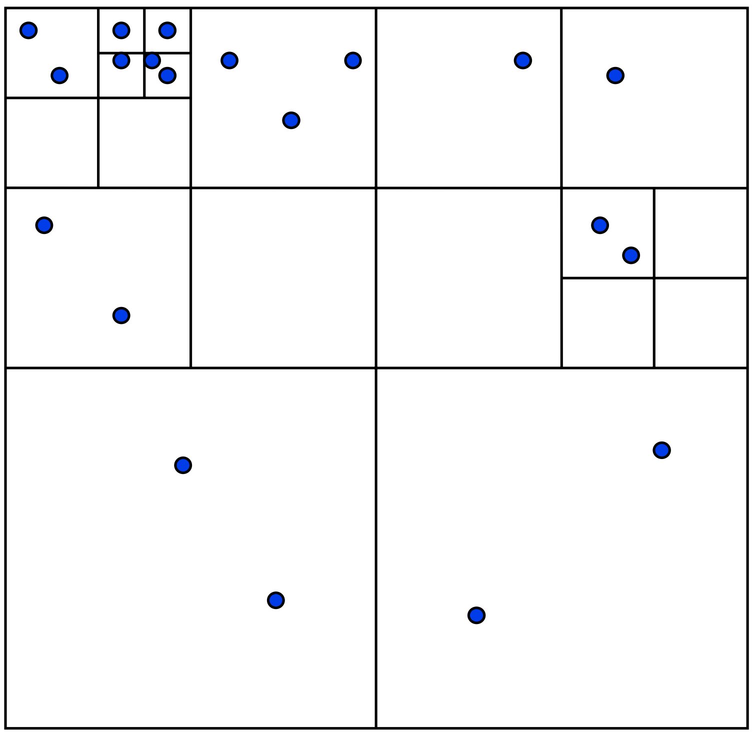
Parallelism in Nearby Forces

- Nearby forces require interaction and therefore communication.
- Force may depend on other nearby particles:
  - Example: collisions.
  - simplest algorithm is O(n^2): look at all pairs to see if they collide.
- Usual parallel model is domain decomposition of physical domain:
  - O(n^2/p) particles per processor if evenly distributed.
- Challenge 1: interactions of particles near processor boundary:
  - need to communicate particles near boundary to neighboring processors.
  - surface to volume effect means low communication.
  - Which communicates less: squares (as below) or slabs?
- Challenge 2: load imbalance, if particles cluster:
  - galaxies, electrons hitting a device wall.



Load balance via Tree Decomposition

- To reduce load imbalance, divide space unevenly.
- Each region contains roughly equal number of particles.
- Quad-tree in 2D, oct-tree in 3D.



Example: each square contains at most 3 particles

Parallelism in Far-Field Forces

° **Far-field forces involve all-to-all interaction and therefore communication.**

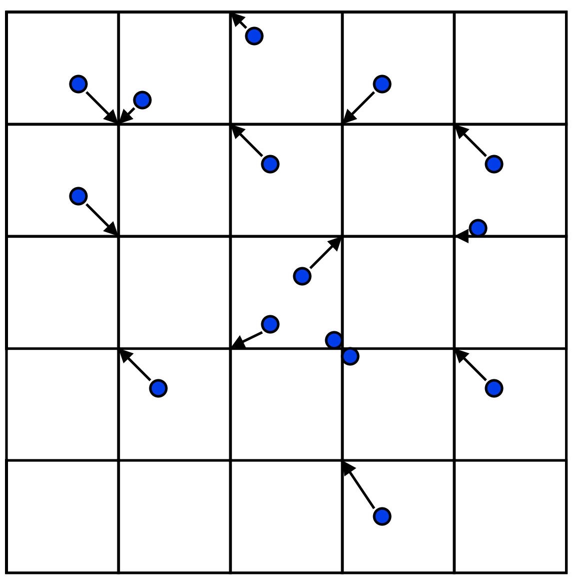
° **Force depends on all other particles:**

- **Example: gravity.**
- **Simplest algorithm is O(n^2) as in S&F 2, 4, 5.**
- **Just decomposing space does not help since every particle apparently needs to "visit" every other particle.**

° **Use more clever algorithms to beat O(n^2).**

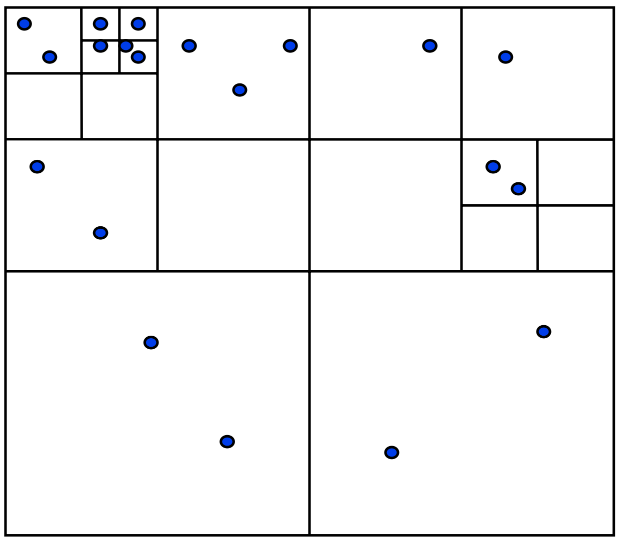
Far-field forces: Particle-Mesh Methods

- **Superimpose a regular mesh.**
- **“Move” particles to nearest grid point.**
- **Exploit fact that the far-field force satisfies a PDE that is easy to solve on a regular mesh:**
  - **FFT, multigrid.**
  - **Wait for future lecture.**
- **Accuracy depends on the fineness of the grid is and the uniformity of the particle distribution.**



Far-field forces: Tree Decomposition

- Based on approximation.
- O(n log n) or O(n) instead of O(n^2).
- Forces from group of far-away particles “simplifies” -- resembles a single large particle.
- Use tree; each node contains an approximation of descendants.
- Several Algorithms
  - Barnes-Hut.
  - Fast multipole method (FMM) of Greengard/Rohklin.
  - Anderson.
  - Later lectures.



Lumped Systems
ODEs

System of Lumped Variables

- **Many systems are approximated by**
  - **System of “lumped” variables.**
  - **Each depends on continuous parameter (usually time).**
- **Example -- circuit:**
  - **approximate as graph.**
  - **wires are edges.**
  - **nodes are connections between 2 or more wires.**
  - **each edge has resistor, capacitor, inductor or voltage source.**
  - **system is “lumped” because we are not computing the voltage/current at every point in space along a wire, just endpoints.**
  - **Variables related by Ohm’s Law, Kirchoff’s Laws, etc.**
- **Forms a system of ordinary differential equations (ODEs).**

Circuit Example

- State of the system is represented by
  - v_n(t) node voltages
  - i_b(t) branch currents
  - v_b(t) branch voltages

  all at time t

- Equations include
  - Kirchoff's current
  - Kirchoff's voltage
  - Ohm's law
  - Capacitance
  - Inductance

$$\begin{pmatrix} 0 & A & 0 \\ A' & 0 & -I \\ 0 & R & -I \\ 0 & -I & C*d/dt \\ 0 & L*d/dt & I \end{pmatrix} * \begin{pmatrix} v\_n \\ i\_b \\ v\_b \end{pmatrix} = \begin{pmatrix} 0 \\ S \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Write as single large system of ODEs (possibly with constraints).

Systems of Lumped Variables

- **Another example is structural analysis in civil engineering:**
  - **Variables are displacement of points in a building.**
  - **Newton's and Hook's (spring) laws apply.**
  - **Static modeling: exert force and determine displacement.**
  - **Dynamic modeling: apply continuous force (earthquake).**
- **The system in these case (and many) will be sparse:**
  - **i.e., most array elements are 0.**
  - **neither store nor compute on these 0's.**

Solving ODEs

- **Explicit methods to compute solution(t):**
  - **Example: Euler's method.**
  - **Simple algorithm: sparse matrix vector multiply.**
  - **May need to take very small time steps, especially if system is stiff (i.e. can change rapidly).**
- **Implicit methods to compute solution(t):**
  - **Example: Backward Euler's Method.**
  - **Larger time steps, especially for stiff problems.**
  - **More difficult algorithm: solve a sparse linear system.**
- **Computing modes of vibration:**
  - **Finding eigenvalues and eigenvectors.**
  - **Example: do resonant modes of building match earthquake vibrations?**
- **All these reduce to sparse matrix problems:**
  - **Explicit: sparse matrix-vector multiplication.**
  - **Implicit: solve a sparse linear system.**
    - **direct solvers (Gaussian elimination).**
    - **iterative solvers (use sparse matrix-vector multiplication).**
  - **Eigenvalue/vector algorithms may also be explicit or implicit.**

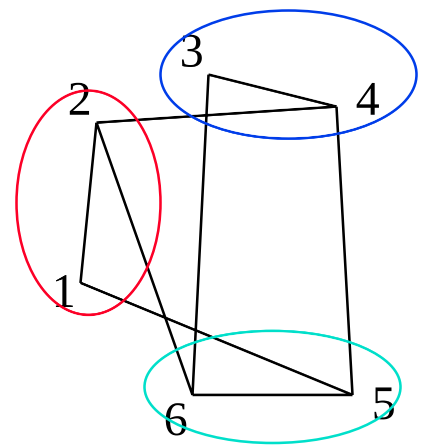
Parallelism in Sparse Matrix-vector multiplication

- **y = A*x, where A is sparse and n x n**
- **Questions:**
  - **which processors store**
    - **y[i], x[i], and A[i,j]**
  - **which processors compute**
    - **x[i] = sum from 1 to n or A[i,j] * x[j]**
- **Graph partitioning:**
  - **Partition index set {1,…,n} = N1 u N2 u … u Np.**
  - **for all i in Nk, store y[i], x[i], and row i of A on processor k.**
  - **Processor k computes its own y[i].**
- **Constraints:**
  - **balance load.**
  - **balance storage.**
  - **minimize communication.**

Graph Partitioning and Sparse Matrices

° **Relationship between matrix and graph**

	1	2	3	4	5	6
1	1	1			1	
2	1	1		1		1
3			1	1		1
4		1	1	1	1	
5	1			1	1	1
6		1	1		1	1



° **A "good" partition of the graph has**

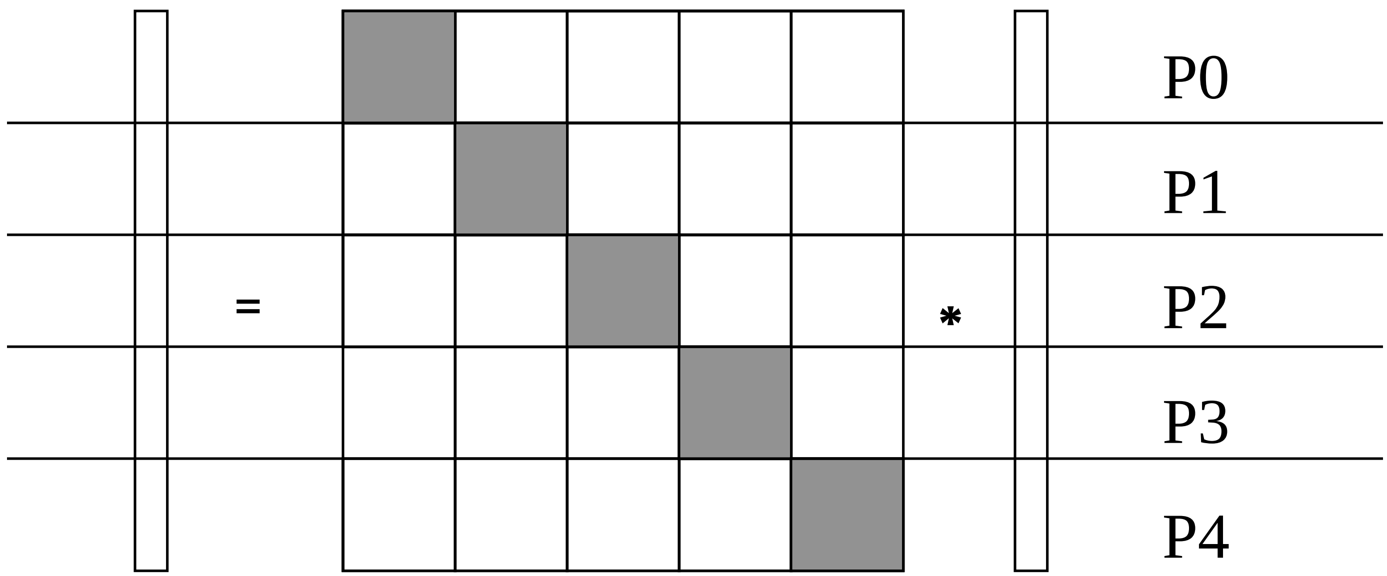
- **equal number of (weighted) nodes in each part (load balance).**
- **minimum number of edges crossing between (minimize communication).**

° **Generally reorder the rows/columns of the matrix by placing all the nodes in one partition together.**

More on Matrix Reordering via Graph Partitioning

° **Goal is to reorder rows and columns to**
  - **improve load balance.**
  - **decrease communication.**

° **"Ideal" matrix structure for parallelism: (nearly) block diagonal:**
  - **p (number of processors) blocks.**
  - **few non-zeros outside these blocks, since these require communication.**



What about implicit methods and eigenproblems?

° **Direct methods (Gaussian elimination)**
  - **future lectures will consider both dense and sparse cases.**

° **Iterative solvers**
  - **future lectures will discuss several of these.**
  - **most have sparse-matrix-vector multiplication in kernel.**

° **Eigenproblems**
  - **future lectures will discuss dense and sparse cases.**
  - **depends on student interest.**